* **Notice:**

{} means hash table and has key: values;

[] means an ordered list; no such notion as unordered list

**Jason format input file: reverseEngineeringInputData**

* It starts with one field that allow us to introduce comments regarding the file. The first field is “description” to describe the file.
* Number of variables indicated as “numberVariables”. This is a non-zero positive integer number.
* The cardinality of the field for the PDSs. This is indicated as “fieldCardinality”. At the moment this has to be a prime number.
* The input time series data indicated as “timeSeriesData”.
* “timeSeriesData” it is organized as an array of time series data.
* Each one of the time series is described as object “matrix”.
* Each one of the “matrix” objects -that describe an experiment-, we specify:
  + The actual time series data. This is represented as a matrix were columns represent the variables and rows are the ordered time steps.
  + “Name” which refers to the name designated to that experiment
  + “index” this used for Knockout data to specify which variable is knocked out. For example if the 3rd variable is knocked out, then we write the as “index” : [3]. Notice that we can have in one experiment more than one variable knocked out . For example, if variables 4 and 5 are knocked out, then we describe such KO experiment as “index”: [4,5]. When we refer to wildtype data, we simply write an empty“index”: [].

**Jason format output file: reverseEngineeringOutputData**

* It starts with one field that allow us to introduce comments regarding the file. The first field is “description” to describe the file.
* Number of variables “numberVariables”. This is a non-zero positive integer number.
* The cardinality of the field for the PDSs. This is indicated as “fieldCardinality”. At the moment this has to be a prime number.
* The PDS will be described as a hash table of update rules indicated by object “updateRules”. The update rule for the ith variable is represented as “x” followed the index of the variable. For instance “xi”, where i>0.
* For each ith variable “xi” its corresponding update rule will be an array of hashtables [{…}, {…},…, {…}], where each hashtable has three objects:
  + The support variables. Designated as “InputVariables”. This is an array of variables written between quotes [“x1”, “x2”]. It is encouraged to write this array in ascending index order.
  + The different polynomial function from the input support variables. For instance “polynomialFunction”: “x1\*x2+x3^2”
  + “polynomialprobability” refers to the probability of the given polynomial function to occur. This polynomial probability is NOT written between quotes. The polynomial probabilities for a given variable xi should add up to 1.
* “variableWeights”. For each variable xi we specify the probability that a given variable xj appears across the possible polynomial functions for xi. It is written a hashtable where the objects are the different variables “x1”, “x2”, etc. In front of these variables, we write the hastable of the different support variables with their corresponding probability. Example:

“x1”: {“x1”: 1.0, “x2”:0.45}

We list these variable between quotes is increasing index number.

If a given variable does not appear in the hash table it is assumed that the variable probability of such variable is zero. These weights are values from 0 to 1. Notice that the sum of the variable weights for xi doesn’t have to add to 1.